ISRN TELCORDIA--2011-02+PR-0GARAU
Prepared for: Office of Naval Research

Fast Multiscale Algorithms for Information Representation and Fusion

Technical Progress Report No. 2

Devasis Bassu, Principal Investigator

Contract: N00014-10-C-0176

Telcordia Technologies

One Telcordia Drive

Piscataway, NJ 08854-4157

January 2011

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **JAN 2011**	2. REPORT TYPE	3. DATES COVERED **00-00-2011 to 00-00-2011**

4. TITLE AND SUBTITLE **Fast Multiscale Algorithms For Information Representation And Fusion**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Telcordia Technologies,Piscataway,NJ,08854**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

In the second quarter of the work effort, we continued research and development of algorithms based on randomized matrix decompositions. These randomized variants have theoretically proven improvements in computational complexity over existing algorithms. Algorithm designs for computing the Randomized Singular Value Decomposition (SVD) using randomized Fast Fourier Transform projections and the Interpolative Decomposition were completed. Fortran 95 interfaces for reusable randomized SVD routines have been defined and implemented. A survey of currently available optimized libraries for the Basic Linear Algebra Subprograms (BLAS) and Linear Algebra PACKage (LAPACK) interfaces was conducted to guide development. The randomized SVD implementation uses these libraries via standardized interfaces to make optimal use of hardware resources (e.g., multiple cores, CPU cache) in addition to using the OpenMP standard (for parallel execution of code). Use of these standards enables the code to be built flexibly in a number of ways on various target platforms. Preliminary testing of the software is complete. Additional updating, fine tuning will be based on results from various experiments that will be conducted in the upcoming quarters.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**14**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

1 Abstract

In the second quarter of the work effort, we continued research and development of algorithms based on randomized matrix decompositions. These randomized variants have theoretically proven improvements in computational complexity over existing algorithms. Algorithm designs for computing the Randomized Singular Value Decomposition (SVD) using randomized Fast Fourier Transform projections and the Interpolative Decomposition were completed. Fortran 95 interfaces for reusable randomized SVD routines have been defined and implemented. A survey of currently available optimized libraries for the Basic Linear Algebra Subprograms (BLAS) and Linear Algebra PACKage (LAPACK) interfaces was conducted to guide development. The randomized SVD implementation uses these libraries via standardized interfaces to make optimal use of hardware resources (e.g., multiple cores, CPU cache) in addition to using the OpenMP standard (for parallel execution of code). Use of these standards enables the code to be built flexibly in a number of ways on various target platforms. Preliminary testing of the software is complete. Additional updating, fine tuning will be based on results from various experiments that will be conducted in the upcoming quarters.

The project is currently on track – in the upcoming quarter, we will focus on research and design of the randomized Approximate Nearest Neighbor (ANN) algorithm. The next two quarters will also include testing and conducting experiments for the randomized SVD and ANN algorithms. No problems are currently anticipated.

Table of Contents

2 Summary

The focus for this quarter was on the continued research and development of algorithms based on randomized matrix decompositions including algorithm designs for the Randomized Singular Value Decomposition (SVD) using random Fast Fourier Transform projections and the Interpolative Decomposition. Fortran 95 interfaces for reusable randomized SVD routines have been defined and implemented that allow for a variety of input matrix types (e.g., sparse, dense, complex valued). Further, the algorithms operate in one of two modes – pre-specified fixed rank or automatic rank detection catering to different application needs.

The implementation uses the Basic Linear Algebra Subprograms (BLAS) and Linear Algebra PACKage (LAPACK) standards apart from the OpenMP standard (for parallel execution on multi-core/multiple CPUs). Preliminary tests were extremely promising – especially for large matrices (both sparse and dense) with low ranks. Further updates and fine-tuning will be based on testing and experiments conducted in the upcoming quarters.

The project is currently on track – we will focus on research and design of the randomized Approximate Nearest Neighbor (ANN) algorithms in the next quarter. No problems are currently anticipated.

3 Introduction

The primary project effort over the last two quarters focused on the design and development of PCA-type schemes and fast regression algorithms based on randomized matrix decompositions – in particular, the Randomized Singular Value Decomposition (SVD) algorithms based on randomized Fast Fourier Transform projections and the Interpolative Decomposition (see [1] [6][7][6][7]). The algorithms have been implemented as reusable routines in Fortran 95 with well-defined interfaces. The implementation uses the standard Basic Linear Algebra Subroutines (BLAS) [8] interface, the Linear Algebra PACK (LAPACK) [9] interface and the OpenMP API [15]. This provides flexibility in terms of compiling the software on a variety of target platforms; exploiting availability of optimized BLAS/LAPACK libraries (see [12][13][12][13]) and availability of multiple cores/CPUs. Further, the implementation caters to a wide variety of input matrices (e.g., sparse, dense, complex-valued) and different types of applications (e.g., pre-specified fixed rank computation, automatic rank detection).

Preliminary tests have been conducted – additional testing and experimentation will be carried out in the upcoming two quarters resulting in possible improvement and fine-tuning of the software.

4 Methods, Assumptions and Procedures

4.1 Randomized Singular Value Decomposition Algorithm

Our implementation of the randomized SVD involves the use of randomized projections using the Fast Fourier Transform and the Interpolative Decomposition (see [4][5][6][7] for details).

The algorithm may be broadly decomposed into three steps – a) a randomization step where the columns of the input matrix are subject to a random projection (to a lower dimensional space) followed by a pivoted QR decomposition to identify the rank and the pivot columns, b) construction of the interpolative decomposition (ID) using the corresponding pivot columns of the original matrix, and finally c) a dense SVD of a smaller matrix and subsequent construction of the SVD of the original matrix. The design considerations for each step are described in detail below.

Note: Assume the input matrix $A_{m,n}$ to be of size m rows and n columns. For all three steps, BLAS operations are used wherever appropriate.

4.1.1 Randomization Step

In this step, we apply a random projection matrix $\varphi_{l,m}$ to each column of A. While this is the most expensive step in the algorithm, it is easily parallelizable since the random projection may be applied to each column independently. Various elements of the random projection (permutations, Givens rotations, FFT coefficients) may be pre-computed upfront.

The matrix $\varphi_{l,m}$ in our implementation comprises a) a sequence of alternating random row permutations and a chain of random Givens rotations, b) a random permutation followed by FFT, c) another random permutation followed by a chain of random Givens rotations, and d) a final projection selecting the top l rows. The size of the sequence is set to a default of 6 based on test results and is also exposed as an optional parameter in the Fortran 95. Note that the FFT computation is dominant in the application of the random projection to columns of A.

As part of the randomized SVD algorithm, for a suitable rank k, l needs to be set to $k + r$. Again, r is set a default of 20 based on test results and is also exposed as an optional parameter in the Fortran 95 interface.

We use our own pivoted QR algorithm that uses a re-orthogonalization strategy at each step to avoid significant loss of numerical accuracy. The pivoted QR algorithms are also exposed via Fortran 95 interfaces and may be reused independent of the randomized SVD routines.

4.1.2 ID Step

To compute the interpolative decomposition, a minimizing least-squares solution is required. The LAPACK routine xGELS is used for this purpose.

4.1.3 SVD Step

In the last step, we use the LAPACK routines xGESDD and xGESVD to compute the SVD of the smaller matrix. We also provide Fortran 95 interfaces to compute the “regular non-randomized” SVD of a matrix using these routines. In particular, these interfaces hide the details of creating appropriate work arrays required for computation from the end-user.

It should be noted that the routine xGESDD while known to compute singular vectors significantly faster than xGESVD may fail to converge for certain input matrices. Our Fortran 95 interface implementation automatically detects such failures and reverts to the slower but guaranteed xGESVD routine.

4.2 Design Considerations

The software has been designed to operate flexibly on a variety of target machines/environments as well as provide an easy-to-use interface for the end-user. To meet the latter requirement, Fortran 95 was chosen since amongst other useful features, it provides dynamic memory allocation permitting the software developer to hide details of allocating memory for required work arrays from the end-user. The onerous task of having to manage work arrays should be quite familiar to the LAPACK user.

The software is built using the standard BLAS and LAPACK interfaces. Additionally, it also used the OpenMP specification for parallelization of various code sections. However, the minimal requirements for compilation are a

- Fortran compiler compliant with the Fortran 95 standard, and
- BLAS and LAPACK libraries (base implementations are available on nearly every platform)

The BLAS and LAPACK libraries may be substituted by optimized equivalents if available for the target machine. Based on our survey of such available libraries, we have selected ATLAS ([11]) since it is used in various other commercial and non-commercial products, comes with source code and is fairly easy to build on our test machines.

Additionally, if the compiler supports the OpenMP specification, the compiled code has additional parallelization support for the target machine. The popular Fortran compiler *gfortran* from the GNU Compiler Collection (GCC) fully supports the OpenMP specification (OpenMP v3.0 as of GCC v4.4).

4.3 Fortran 95 Interface for Randomized SVD

The Fortran 95 interfaces for computing the randomized SVD use the LAPACK naming convention. The xGERSVDF routine computes the randomized SVD for any matrix specified as a function which returns the matrix value for the specified row and column. The first character x may be one of S, D, C, Z to indicate single/double precision real or single/double precision complex.

SUBROUTINE xGERSVDF(nRows, nCols, matFun, U, D, V, kEst, errNorm, retCode, procTimes, k, blockSize, randSize, randIters, errTol)

Description: Constructs a nearly optimal rank-k approximation U*D*V' to the matrix (specified as the function matFun). The low-rank approximation U*D*V' is in the form of an SVD in the sense that the columns of U are orthonormal, as are the columns of V, the entries of D are all nonnegative, and the only nonzero entries of D appear in non-increasing order on its diagonal (the routine provides only the diagonal entries).

Input Arguments
nRows :: INTEGER - number of rows
nCols :: INTEGER - number of columns
matFun :: EXTERNAL function of type x - matrix specified as a value function of (row,col)
Note: The arguments row and col range from 1 to nRows and nCols respectively.
k :: INTEGER - desired rank (used as an upper limit while computing the SVD) [*optional*]
Note: If k is not provided, the algorithm computes the rank automatically. In the case where k is provided and (k + randLevel) > MIN(nRows, nCols), a non-randomized SVD is computed. Also, in the case where k is not provided, if (estimated rank + randLevel) > MIN(nRows, nCols), a non-ramdomized SVD is computed.
blockSize :: INTEGER - block size for parts of the algorithm that use blocking. Optimal values are typically machine dependent. (defaults to 32) [*optional*]
randSize :: INTEGER - controls the accuracy of the approximation (defaults to 20) [*optional*]
randIters :: INTEGER - controls the amount of randomization (defaults to 6) [*optional*]
Note: In general, do NOT override these parameters without proper understanding of the algorithm.
errTol :: REAL - precision used for error tolerance (defaults to machine precision) [*optional*]

Output Arguments
kEst :: INTEGER - final estimate of rank for the approximation (not to exceed the desired rank k)
U :: DIMENSION(:,:), POINTER of type x - singular vectors corresponding to column space of size nRows,kEst
D :: DIMENSION(:), POINTER of type x - singular values of size kEst
V :: DIMENSION(:,:), POINTER of type x - singular vectors corresponding to row space of size nCols, kEst
errNorm :: type x - norm estimate of approximation error ||mat - U*S*V'||
retCode :: INTEGER - return code
procTimes :: REAL, DIMENSION(4) - wall clock processing times in seconds for matrix randomization, interpolative decomposition, generation of the SVD and total time (in that order). If a non-randomized SVD is performed, only the last value is meaningful.
IMPORTANT U, S, V must be DEALLOCATED after use to avoid memory leaks.

Return Code Values
0 – Success
1 - Invalid arguments

2 - Out of memory
3 - Failed to obtain least-squares solution
4 - LAPACK xGESDD algorithm failed to converge
5 - LAPACK xGESDD algorithm has invalid arguments
6 - LAPACK xGESDD algorithm failed to return optimal work size

Note: The above interface may be subject to slight modifications based on the testing and experimentation carried out in the next two quarters.

4.4 Deliverables / Milestones

Date	Deliverables / Milestones	Status
Oct 2010	Progress report for period 1, 1st quarter	✓
Jan 2011	Progress report for period 1, 2nd quarter / complete randomized matrix decompositions task	✓
Apr 2011	Progress report for period 1, 3rd quarter / complete approximate nearest neighbors task	
Jul 2011	Progress report for period 1, 4th quarter / complete experiments – part 1	
Oct 2011	Progress report for period 2, 1st quarter	
Jan 2012	Progress report for period 2, 2nd quarter / complete multiscale SVD task	
Apr 2012	Progress report for period 2, 3rd quarter	
Jul 2012	Progress report for period 2, 4th quarter / complete experiments – part 2	
Oct 2012	Progress report for period 3, 1st quarter	
Jan 2013	Progress report for period 3, 2nd quarter / complete multiscale Heat Kernel task	
Apr 2013	Progress report for period 3, 3rd quarter	
Jul 2013	Final project report + software + documentation on CDROM / complete experiments – part 3	

5 Results and Discussion

We present some results from our preliminary testing of the randomized Singular Value Decomposition algorithm to highlight the

a) accuracy of the randomized SVD, and
b) speed of computing the randomized SVD

with the baseline of computing the non-randomized SVD for the same input matrix.

5.1 Test Setup

Several double precision random square dense matrices of size 4096 were constructed with ranks set to 50, 100, 150, 200, 250 and 300. The following was computed for each matrix A.

a) Randomized SVD of $A \cong U_R \cdot S_R \cdot V_R^*$
b) Frobenius norm of the approximation error $\varepsilon_{RSVD} = \|A - U_R \cdot S_R \cdot V_R^*\|_F$
c) Total time T_R and individual times for randomization step T_{RND}, ID step T_{ID} and final SVD step T_{SVD}
d) Non-randomized SVD of $A \cong U \cdot S \cdot V^*$
e) Frobenius norm of the approximation error $\varepsilon_{SVD} = \|A - U \cdot S \cdot V^*\|_F$
f) Total time T
g) Maximum relative error for the randomized estimate of singular values using the non-randomized estimate as the true value $rel\text{-}\varepsilon_S = max_i \left|\frac{S(i) - S_R(i)}{S(i)}\right|$

Further, the singular vectors in all cases were checked to be orthonormal up to a given precision. The DGESDD routine from LAPACK was used to compute all non-randomized SVDs. The DGERSVDF routine was used to both detect the rank automatically and compute the SVD (the rank was correctly detected in all cases).

All tests were carried out on a machine with four Intel Xeon 2.4GHz 6-core CPUs (total of 24 cores) with a total of 64 GB main memory. The OS was Ubuntu 9.10 64-bit SMP with GCC v4.4.1 (supporting OpenMP v3.0). The Ubuntu packaged version of ATLAS v3.6.0 (single-threaded) was used as a baseline. ATLAS v3.8.3 with full LAPACK was compiled on the machine to provide optimized multi-threaded BLAS/LAPACK libraries.

The following reports results for three test cases – a) the base case with no optimizations or parallelization, b) second case with optimized BLAS/LAPACK libraries but without OpenMP compiled support, and c) third case with optimized BLAS/LAPACK libraries and OpenMP compiled support.

5.2 Case 1: No optimized BLAS/LAPACK, no OpenMP support

Rank	T (secs)	T_R (secs) T_{RND} T_{ID} T_{SVD}	ε_{SVD}	ε_{RSVD}	$rel\text{-}\varepsilon_S$
50	289.51	13.91 (13.72 + 0.05 + 0.14)	2.54E-010	6.67E-011	8.80E-015
100	289.98	19.09 (18.43 + 0.12 + 0.54)	5.34E-010	9.91E-010	8.88E-015
150	289.78	30.05 (28.88 + 0.17 + 1.00)	4.99E-010	1.18E-009	6.68E-014
200	290.11	36.58 (34.56 + 0.39 + 1.63)	4.92E-010	2.16E-009	2.26E-014
250	290.97	37.26 (34.38 + 0.45 + 2.43)	6.03E-010	1.62E-009	6.82E-014
300	290.75	39.79 (35.81 + 0.53 + 3.45)	7.81E-010	3.85E-009	1.53E-013

5.3 Case 2: Optimized BLAS/LAPACK, no OpenMP support

Rank	T (secs)	T_R (secs) T_{RND} T_{ID} T_{SVD}	ε_{SVD}	ε_{RSVD}	$rel\text{-}\varepsilon_S$
50	204.18	8.45 (8.33 + 0.04 + 0.08)	3.52E-010	1.43E-010	1.35E-014
100	201.04	28.83 (28.46 + 0.08 + 0.29)	9.82E-010	3.53E-010	1.68E-014
150	200.85	30.42 (29.69 + 0.10 + 0.63)	9.48E-010	1.90E-009	7.99E-014
200	201.06	36.14 (34.88 + 0.19 + 1.07)	8.43E-010	7.34E-010	2.37E-014
250	200.88	36.22 (34.41 + 0.22 + 1.59)	9.91E-010	1.35E-009	5.74E-014
300	201.46	38.66 (36.13 + 0.26 + 2.27)	1.43E-009	3.23E-009	1.65E-013

5.4 Case 3: Optimized BLAS/LAPACK, OpenMP support

Rank	T (secs)	T_R (secs) T_{RND} T_{ID} T_{SVD}	ε_{SVD}	ε_{RSVD}	$rel\text{-}\varepsilon_S$
50	203.82	8.42 (7.71 + 0.49 + 0.22)	4.47E-010	1.59E-010	1.37E-014
100	201.31	12.00 (11.17 + 0.32 + 0.51)	6.62E-010	7.70E-010	1.40E-014
150	201.39	11.74 (10.90 + 0.17 + 0.67)	7.84E-010	1.83E-009	6.24E-014
200	200.87	16.71 (15.32 + 0.28 + 1.11)	8.02E-010	7.19E-010	2.51E-014
250	201.12	17.19 (15.25 + 0.30 + 1.64)	9.87E-010	1.23E-009	5.72E-014
300	201.10	16.82 (14.15 + 0.35 + 2.32)	1.30E-009	3.20E-009	1.17E-013

The results for ε_{RSVD} indicate stable behavior and overall good approximation. Also, the numbers for $rel\text{-}\varepsilon_S$ show strong agreement of the singular values with those computed using the LAPACK DGESDD routine.

The results are in conformance with expected speedups (or lack thereof) for compute times. In particular,

- use of optimized BLAS/LAPACK libraries help speed up non-randomized SVD computation significantly. However, even though there is some gain in the third step (SVD computation) of the randomized SVD algorithm, it is dominated by the timings for the first step.
- parallelization of the first step (random projection) provides significant gains for the randomized SVD algorithm

In summary, use of both optimized libraries and parallelization provide significant boosts in terms of compute times.

It should be pointed out that the comparison between the totals times between the non-randomized and randomized versions of the SVD is strictly not fair as the non-randomized LAPACK routine computes the full singular vectors (not the economic or compact version). Even so, the speedup factor for case 3 ranges from 24 (for rank 50) to 12 (for rank 300) which is highly significant!

6 Conclusions

The project is on track with the completion of design and implementation of the first algorithm (Randomized Singular Value Decomposition). We will continue with further testing and application of the algorithm to a variety of real-world data sets. Research and development of the second algorithm (Randomized Approximate Nearest Neighbors) is already underway; we will start on algorithm design next quarter.

No problems are currently anticipated.

7 References

[1] G.H. Golub, C.F. Van Loan, *Matrix Computations 3rd Ed.*, Johns Hopkins University Press, Baltimore, 1996.
[2] G.M. Stewart, *Matrix Algorithms Volume I: Basic Decompositions*, SIAM, 1998.
[3] G.M. Stewart, *Matrix Algorithms Volume II: Eigensystems*, SIAM 2001.
[4] E. Liberty, F. Woolfe, P.G. Martinsson, V. Rokhlin, M. Tygert, *Randomized Algorithms for the Low-Rank Approximation of Matrices* PNAS, V. 104, No. 51, pp. 20167-20172, 2007.
[5] E. Liberty, F. Woolfe, P.G. Martinsson, V. Rokhlin, M. Tygert, *A Fast Randomized Algorithm for the Approximation of Matrices*, Applied and Computational Harmonic Analysis, v. 25, pp.335-366, 2008.
[6] V. Rokhlin and M. Tygert, *Fast algorithms for spherical harmonic expansions*, SIAM Journal on Scientific Computing, 27 (6): 1903-1928, 2006.
[7] V. Rokhlin, M. Tygert, *A fast Randomized Algorithm for the Overdetermined Linear Least Squares Regression*, PNAS, v. 105, No. 36, pp. 13212-13217, 2008.
[8] Basic Linear Algebra Subprograms (BLAS), URL:http://www.netlib.org/blas
[9] Linear Algebra PACK, URL:http://www.netlib.org/lapack
[10] GotoBLAS2, URL:http://www.tacc.utexas.edu/tacc-projects/gotoblas2/
[11] Automatically Tuned Linear Atlas Subroutines (ATLAS), URL:http://www.netlib.org/atlas
[12] Intel Math Kernel Library, URL:http://software.intel.com/en-us/articles/intel-mkl
[13] AMD Core Math Library (ACML), URL:http://developer.amd.com/cpu/Libraries/acml/Pages/default.aspx
[14] NIST Matrix Market, URL:http://math.nist.gov/MatrixMarket
[15] The OpenMP API Specification for Parallel Programming, URL:http://www.openmp.org